Scheduling to Minimize Energy and Flow Time in Broadcast Scheduling

Benjamin Moseley*

Abstract

In this paper we initiate the study of minimizing power consumption in the broadcast scheduling model. In this setting there is a wireless transmitter. Over time requests arrive at the transmitter for pages of information. Multiple requests may be for the same page. When a page is transmitted, all requests for that page receive the transmission simulteneously. The speed the transmitter sends data at can be dynamically scaled to conserve energy. We consider the problem of minimizing flow time plus energy, the most popular scheduling metric considered in the standard scheduling model when the scheduler is energy aware. We will assume that the power consumed is modeled by an *arbitrary* convex function. For this problem there is a $\Omega(n)$ lower bound. Due to the lower bound, we consider the resource augmentation model of Gupta et al. [28]. Using resource augmentation, we give a *scalable* algorithm. Our result also gives a scalable *non-clairvoyant* algorithm for minimizing weighted flow time plus energy in the standard scheduling model.

*Department of Computer Science, University of Illinois, 201 N. Goodwin Ave., Urbana, IL 61801. bmosele2@illinois.edu. Partially supported by NSF grant CNS-0721899.

1 Introduction

Wireless transmitters can typically be utilized in a variety of ways. This presents the designer with tradeoffs between power, signal range, bandwidth, cost, ect. In this paper we consider the trade off between power and performance. Reducing the energy consumption is of importance in many systems. For example, in ad hoc wireless networks a typical transmitter is battery operated and, therefore, energy conservation is critical. There are two modes of a wireless transmitter where power can be saved. (1) During idle times (2) During transmit times. Work on the first mode focuses on putting the system to sleep when not in use [17, 2]. By appropriately putting the system to sleep, energy consumption can be drastically reduced. Another way the transmitter can reduce power is scaling the speed of the wireless transmission [31]. By using more power a signal can be sent at a faster rate or, to save power, the signal can be sent at a slower rate. We focus on reducing energy in the second mode. Reducing energy by changing the transmission speed is naturally related to the well studied model of speed scaling in scheduling theory. However, now that we are in interested in wireless networks, this motivates a generalization of the standard speed scaling model.

Companies such as IBM and AMD are producing processors whose speed can be dynamically scaled by the operating system. Now an operating systems can control the power consumed in the system by scaling the processor speed. To model this is scheduling theory, it is assumed that there is a power function P where $P(s)$ is the power use when running the processor at speed s. A scheduler now not only chooses the job to schedule, but also the speed used to process the job. In other words, a scheduler determines a job selection policy and an energy policy. This is known as the speed scaling scheduling model.

The standard speed scaling model is similar to our setting, except that we are interested in speed scaling in wireless networks. Like the speed scaling setting, we assume there is a power function P where $P(s)$ is the power used when transmitting at speed s. To model the wireless network, we will assume a well-studied model known as the broadcast model. In a broadcast network, there is a single server and n pages of information are stored at the server. Over time clients send requests for pages to the server. Multiple clients could be interested in the same page of information. When the sever broadcasts a page p, all outstanding requests for that page are simultaneously satisfied. This is how the broadcast model differs from the standard scheduling model. In the standard model, a each request (job) is for a unique page and broadcasting (processing) a page satisfies exactly one request. In the *online* setting, the server is not aware of a request until it arrives to the system. The broadcast model is motivated by applications in multicast systems and in wireless and LAN networks [38, 1, 3, 29]. Along with practical interest, the broadcast model has been studied extensively in scheduling literature [10, 3, 1, 11, 29, 26]. Similar models have been considered in queuing theory and the stochastic scheduling model [19, 18, 36, 37].

The goal of the scheduler is to satisfy the requests in an order which optimizes a quality of service metric. Naturally, this metric depends on the needs of the system. When speed scaling is allowed, the server has a dual objective. One is to minimize the power usage and the other is to optimize the quality of service the clients receive. The most popular metric in speed scaling literature is minimizing a linear combination of flow time and total energy [4, 9, 12, 28, 5]. The flow time[1] of a request is the amount of time the server takes to satisfy the request. Let F denote the total flow time of a schedule over all requests and let E denote the energy consumption of the schedule. The scheduler focuses on minimizing $G = F + E$. This has a natural interpretation. Say the system is willing to spend one unit of energy to reduce ρ units of flow time. For example, a system designer may be able to justify spending 1 erg of energy to decrease the flow time by $\rho = 10$ micro seconds. The system designer would then desire a schedule that minimizes $F + 10E$. By scaling the units of energy and flow time, we can assume that $\rho = 1$. The main contribution of this work is to initiate the study of energy in the broadcast model. We focus on the the problem of minimizing flow time plus energy in the broadcast setting. Besides practical interest in the model, we believe this problem is of

[1] Flow time is also known as waiting time or response time

theoretical interest as it is a natural extension of previous work.

Results: In this paper we give an algorithm for minimizing total flow time plus energy in the online broadcast setting where the power function is an *arbitrary* convex function. There is a $\Omega(n)$ lower bound on the problem of minimizing flow time in broadcast scheduling when all broadcasts are sent at a fixed rate and energy is not included in the objective [33]. Since the power function is arbitrary, this lower bound also extends to the problem of minimizing flow time plus energy. Thus we will resort to resource augmentation [32]. The resource augmentation model we consider is the same as that introduced recently in [28]. Here, if our algorithm is given $1+\epsilon$ speed advantage over the adversary, then our algorithm uses power $P(\gamma)$ when broadcasting at a rate of $(1+\epsilon)\gamma$. The rest of the paper will be devoted to proving the following theorem.

Theorem 1.1. *There exists an algorithm that with $1+\epsilon$ resource augmentation is $O(\frac{1}{\epsilon^3})$-competitive for minimizing total flow time plus energy in broadcast scheduling with an arbitrary power function where $0 < \epsilon \leq 1$.*

Our algorithm is called *scalable* since the minimum amount of resource augmentation is used to be $O(1)$-competitive. Given the strong lower bound, this is the best result that can be shown in the worst case setting up to a constant in the competitive ratio. Broadcast scheduling is a generalization of the standard scheduling model. The standard model can be interpreted as each request being for a unique page. In fact, energy plus flow time in the broadcast scheduling strictly generalizes weighted flow time plus energy in the standard speed scaling model. Thus, our result also gives a scalable algorithm for minimizing energy plus weighted flow time in standard scheduling setting. In the standard model, an algorithm is said to be *non-clairvoyant* if it is online and does not know the processing time of a job. The algorithm we introduce is non-clairvoyant when interpreted in standard scheduling model.

Theorem 1.2. *There exists a non-clairvoyant algorithm that with $1+\epsilon$ resource augmentation is $O(\frac{1}{\epsilon^3})$-competitive for weighted flow time plus energy in the standard speed scaling model.*

Before this work, no non-trivial non-clairvoyant algorithm was known for (un)weighted flow time plus energy when the power function can be arbitrary in the standard speed scaling setting. However, it was known that no non-clairvoyant algorithm can be $O(1)$-competitive for this problem [12]. Thus, resource augmentation is necessary for an algorithm to be $O(1)$-competitive.

Relation to Previous work: In the standard scheduling setting, speed scaling has traditionally used power functions of the form $P(s) = s^\alpha$ for some $\alpha > 0$. These power functions were motivated by the fact that the power used by CMOS based processors is approximately s^3. Recently, modeling the power function as an arbitrary convex was suggested to capture the power consumption of more complex systems [8]. As mentioned, in this work, we adopt this new general model.

There is a large amount of literature on scheduling in the broadcast model. The most popular scheduling metric considered is minimizing average flow time. Recently, it was shown that there exists an online scheduler that is a scalable algorithm for minimizing the total flow time of a schedule when broadcasts are sent at a fixed rate and energy is not considered in the objective [30, 8]. This work builds on the ideas and techniques given in [8]. In particular, this paper uses the algorithm of [8] with the addition feature of being power aware. Since the power function considered in this paper is an arbitrary convex function, our work generalizes these results.

Previous work on Flow-time in Broadcast Scheduling: Minimizing flow time (without energy) in broadcast scheduling where broadcasts are sent at a fixed speed has a rich history beginning with the seminal work

of [33, 11]. In [33], Kalyanasundaram et al. showed that no online deterministic algorithm can be $\Omega(n)$-competitive. This has since been extended to show that no randomized online algorithm can be $\Omega(\sqrt{n})$-competitive [6]. Due to these strong lower bounds, most previous work has focused on analyzing algorithms in a resource augmentation model [32]. In this resource augmentation analysis, the online algorithm is given s-speed and is compared to a 1-speed adversary. An algorithm is said to be s-speed c-competitive for some objective function if the algorithm's objective when given s speed is within a c factor of the optimal solution's objective schedule given 1-speed for every request sequence. An algorithm that is $(1+\epsilon)$-speed $O(1)$-competitive is called scalable where $0 < \epsilon \leq 1$ since it is $O(1)$-competitive when given the minimum advantage over the adversary.

In the offline setting, the problem was shown to be NP-Hard [25, 14]. The first $O(1)$-speed $O(1)$-approximation was found in [33]. The best positive result in the offline setting using resource augmentation is a $(1+\epsilon)$-speed $O(1)$-approximation [6]. Without resource augmentation, [6] gave a $O(\sqrt{n})$-approximation. This has since been improved to a $O(\log^2 n/\log\log n)$ approximation [7]. It is long standing open question whether or not this problem admits a $O(1)$-approximation.

It has been difficult to find competitive online algorithms for broadcast scheduling. The first online algorithm was given by Edmonds and Pruhs in [22]. This algorithm was shown to be $(4+\epsilon)$-speed $O(1)$-competitive algorithm via a reduction to a different scheduling problem known as arbitrary speed up curves. This reduction takes a s-speed c-competitive algorithm for the speed up curves setting and converts it into a $(2s)$-speed c-competitive algorithm for the broadcast setting. In [23] Edmonds and Pruhs showed that a natural algorithm Longest-Wait-Fist (LWF) is 6-speed $O(1)$-competitive via a global charging argument. The analysis of LWF has been improved to show that the algorithm is $(3.4+\epsilon)$-speed $O(1)$-competitive [15].

Later, Edmonds and Pruhs [24] gave an algorithm LAPS that is scalable in the arbitrary speed up curves setting. Using the reduction [22] this gives a $(2+\epsilon)$-speed $O(1)$-competitive algorithm for the broadcast setting. It was a long standing problem whether or not there existed a scalable online algorithm in the broadcast model. This question was resolved by Im and Moseley and Bansal et al. by finding scalable algorithms [30, 8]. The reduction of [22] was improved by Bansal et al. [8] to show that a s-speed c-competitive algorithm for the speed up curves setting can be converted into a $s(1+\epsilon)$-speed $O(\frac{c}{\epsilon})$-competitive algorithm for the broadcast setting. This shows that BLAPS, the broadcast version of LAPS, is a scalable algorithm in the broadcast model. Besides improving the reduction, in [8] the algorithm BLAPS was analyzed directly via a potential function analysis. This analysis has since been extended to show an algorithm that is $(2+\epsilon)$-speed $O(1)$-competitive for the ℓ_2-norm of flow time [27]. Recently, a scalable algorithm was found for the ℓ_k-norms of flow time for all $k > 0$ [21].

Previous work on Speed-Scalaing: All of the previous work on broadcast scheduling has assumed that the scheduler broadcasts at some fixed speed. Although speed scaling has not been considered in broadcast scheduling, it has been considered extensively in the standard scheduling model. As mentioned, the standard scheduling model can be interpreted as each request being for a unique page. Recall that is the speed scaling setting, a power function P is given where $P(s)$ is the power used when running the processor at speed s. We first consider the traditional model where $P(s) = s^\alpha$ and $\alpha > 1$. In [35] an efficient algorithm was given for the problem of minimizing flow time offline subject to a bound on the amount of power consumed. This algorithm can be extended to find a schedule that minimizes the flow time plus energy in the offline setting. The problem of minimizing flow time plus energy online was first studied by [4]. [9] showed that the algorithm that runs jobs with power proportional to the number of outstanding requests is 4-competitive for unit work jobs. They also showed the Highest-Density-First algorithm coupled with this power strategy is $O(\frac{\alpha^2}{\log^2 \alpha})$ competitive for weighted flow plus energy. This is since been improved in [34] for unweighted flow plus energy to give a $O(\frac{\alpha}{\log \alpha})$-competitive algorithm. In [12] a $O(\alpha^3)$-competitive *non-clairvoyant*

algorithm was given. Chan et al. in [13] gave a $O(\log m)$-competitive algorithm for the speed-up curves setting when there is m processors.

Recently, [5] introduced the problem of minimizing flow time plus energy with an arbitrary convex power function. Surprisingly, they were able to give an algorithm that is 3-competitive in this general setting for flow plus energy. This resolved a central question on whether an algorithm could be $O(1)$-competitive where the competitive ratio does not depend on α. Gupta et al. gave a scalable algorithm for minimizing weighted flow time plus energy in the case where there are m machines, each machine may have a different power function, and each power function is an arbitrary convex function [28]. As mentioned, in this paper we adopt the assumption that the power function is an arbitrary convex function.

2 Preliminaries

We begin by introducing some notation. There are n pages stored at the server. Each page has a size σ_p. A request for page p is satisfied if it receives σ_p pieces of page p in sequential order. That is, a request receives the transmission of page p from start to finish in that order. This will be further elaborated on later. Notice that by this definition, multiple requests can be satisfied by a single broadcast. Let $J_{p,i}$ denote the ith request for page p. Let $a_{p,i}$ be the time this request arrives to the server. In the online setting, this is the first time the server is aware of the request. Let $f_{p,i}$ be the time the server satisfies request $J_{p,i}$. The total flow time of a schedule is $F = \sum_p \sum_i (f_{p,i} - a_{p,i})$. The following definitions will be useful.

Definition 2.1 (convex function)**.** *A real valued function f is convex if and only if for any $0 \leq \alpha \leq 1$ and any real valued x and y it is the case that $f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha)f(y)$*

Definition 2.2 (concave function)**.** *A real valued function f is concave if and only if for any $0 \leq \alpha \leq 1$ and any real valued x and y it is the case that $f(\alpha x + (1-\alpha)y) \geq \alpha f(x) + (1-\alpha)f(y)$*

Using the definition of a concave function, we can easily show the following proposition.

Proposition 2.3. *For any real valued concave function f where $f(0) = 0$, the following holds*

- *For any positive real number x, $\frac{x-1}{x} \leq \frac{f(x-1)}{f(x)}$*
- *For any positive real number x and any $\alpha \leq 1$, $\alpha f(x) \leq f(\alpha x)$*

We will be given a power function $P : \mathbb{R} \to \mathbb{R}$. The value of $P(s)$ is the power used when the server broadcasts with speed s. In [5, 28] with a small loss in the competitive ratio, it was assumed that power function P satisfies the following conditions,

1. At all speeds, P is defined, continuous and differentiable
2. P(0) = 0
3. P is strictly increasing
4. P is strictly convex
5. P is unbounded

In this paper we adopt these assumptions on P. Notice that the speed could be bounded. If the algorithm is given $(1+\epsilon)$ resource augmentation and the maximum speed is s_{max} then our algorithm's maximum speed is $(1+\epsilon)s_{max}$. Let the function $Q = P^{-1}$. That is, $Q(y)$ is the maximum speed of the processor with a limit of y on the power used. Notice that $Q(0) = 0$ and that Q is concave. Let $s(t)$ denote the speed used at time t. Let $E = \int_{t=0}^{\infty} P(s(t))\mathrm{d}t$ be the total energy used. The goal of the scheduler is to minimize $G = F + E$.

2.1 Fractional Broadcast Scheduling

In *integral* broadcast scheduling (the standard model), a request $J_{p,i}$ for page p is satisfied if it receives each of σ_p pieces of page p in sequential order. That is, a page is divided into pieces. At any time the scheduler decides which piece to broadcast. A request is satisfied if it receives each of the pieces from start to finish, in that order. We now define a different version of the broadcast scheduling problem called *fractional* broadcast scheduling. In this setting, the σ_p unit sized pieces of page p are indistinguishable. Now a request $J_{p,i}$ is satisfied once the server devotes a total of σ_p time units to page p after time $a_{p,i}$. In [8] it was shown that an online algorithm running at a fixed speed that satisfies a request $J_{p,i}$ by time t in a fractional schedule can be converted into a different online algorithm that completes $J_{p,i}$ by time $t + \frac{3}{\epsilon'}(t - a_{p,i}) + \frac{5}{\epsilon'}$ in an integral broadcast schedule. Here, the new algorithm is given an additional ϵ' resource augmentation.

It is not obvious that this generalizes when the server can use varying speeds over time. In Appendix A we extend their result to the speed scaling setting. We prove the following theorem. This theorem may be of independent interest, since it can be used to reduce integral broadcast scheduling to the fractional setting for a variety of objective functions where speeds can vary over time.

Theorem 2.4. *Let S be a fractional broadcast schedule where the server can vary its speed over time and let $0 \leq \epsilon' \leq 1$. The schedule $\mathcal{S}$ can be converted into a schedule $\mathcal{S}'$ using ϵ' resource augmentation such that the schedule $\mathcal{S}'$ satisfies the following properties*

- *The power used by $\mathcal{S}'$ is at most the power used by $\mathcal{S}$.*
- *If a request $J_{p,i}$ is fractionally satisfied at time $f_{p,i}$ under S then this request is integrally satisfied at time $f_{p,i} + \frac{2}{\epsilon'}(f_{p,i} - a_{p,i}) + \frac{5}{\epsilon'}$ under the schedule S'.*
- *If the algorithm that generates S is online then so is the algorithm that generates S'.*

This theorem implies the following corollary.

Corollary 2.5. *An algorithm with $1 + \epsilon$ resource augmentation that is c-competitive for flow time plus energy in fractional broadcast scheduling can be converted into an algorithm that is $\frac{7c}{\epsilon'}$-competitive for integral broadcast scheduling and uses $(1 + \epsilon)(1 + \epsilon')$ resource augmentation where $0 < \epsilon' \leq 1$.*

Due to the previous corollary, we will focus on fractional broadcast scheduling for the rest of this paper. It is important to note that if each request is for a unique page, the fractional broadcast setting is equivalent to the standing scheduling setting. This reduction is not needed for the proof of Theorem 1.2

2.2 The Algorithm

Let $N_a(t)$ contain the unsatisfied requests under our algorithm's schedule at time t. Our algorithm broadcasts at speed $Q(|N_a(t)|)$ at time t. Intuitively, since the flow time of the schedule at time t increases by $|N_a(t)|$, the scheduler can afford to use this much power at time t. Now we define how our algorithm distributes its processing power. Here the algorithm BLAPS is used. This algorithm was introduced in [24, 8]. We will be assuming that our algorithm is given $(1 + 6\epsilon)$ resource augmentation and this implies the total speed our algorithm uses at time t is $(1 + 6\epsilon)Q(|N_a(t)|)$ where $\epsilon \leq \frac{1}{6}$. The algorithm BLAPS takes a parameter $\beta \leq 1$. We will fix $\beta = \epsilon$ later. Let $N'_a(t)$ be the $\beta|N_a(t)|$ requests in $N_a(t)$ with the latest arrival times. At any time t, the algorithm equally distributes its processing power amongst the requests in $N'_a(t)$. That is, for a request $J_{p,i} \in N'_a(t)$ page p is broadcasted at a rate of $x_{p,i}(t) = \frac{(1+6\epsilon)Q(|N_a(t)|)}{\beta|N_a(t)|}$. Notice that there could be multiple requests for page p in $N'_a(t)$. Let $S_p(t)$ be the set of requests for page p in $N'_a(t)$. A page p is broadcasted at a rate of $\sum_{J_{p,i} \in S_p(t)} x_{p,i}(t)$ at time t. We call $x_{p,i}(t)$ the amount $J_{p,i}$ contributes to how much page p is broadcasted. Notice that at any time, the algorithm uses total speed $(1 + 6\epsilon)Q(|N_a(t))$.

For brevity, throughout this paper will will be using the following simplifying assumption. We assume that at each time t, $\beta|N_a(t)|$ is integral. A simple extension of the analysis can be made if this is not the case with the following elaboration. Let $N'_a(t)$ now be the $\lceil \beta|N_a(t)| \rceil$ requests in $N_a(t)$ with latest arrival times. For the $\lfloor \beta|N_a(t)| \rfloor$ requests with latest arrival times in $N'_a(t)$, BLAPS keeps the value of x the same. Let $J_{p,i}$ be the only other request in $N'_a(t)$. For this request, we set $x_{p,i} = (\beta|N_a(t)| - \lfloor \beta|N_a(t)| \rfloor) \frac{(1+\epsilon)Q(|N_a(t)|)}{\beta|N_a(t)|}$. That is, $J_{p,i}$ is given processing power proportional to $(\beta|N_a(t)| - \lfloor \beta|N_a(t)| \rfloor)$, the amount $J_{p,i}$ 'overlaps' the $\beta|N_a(t)|$ latest arriving requests.

3 Analysis

We will be using a potential function argument [20]. Let $\frac{\mathrm{d}}{\mathrm{dt}}G(t)$ be the increase in our algorithm's objective at time t. Likewise, let $\frac{\mathrm{d}}{\mathrm{dt}}G^*(t)$ be the increase in OPT's objective at time t. Notice that $\frac{\mathrm{d}}{\mathrm{dt}}G(t) = 2|N_a(t)|$ because there are $|N_a(t)|$ outstanding requests at time t, which increases the flow time of the schedule by $|N_a(t)|$ and the scheduler uses power $|N_a(t)|$ at time t. Let $G = \int_0^\infty \frac{\mathrm{d}}{\mathrm{dt}}G(t)\mathrm{dt}$ denote our algorithm's total cost and let $G^* = \int_0^\infty \frac{\mathrm{d}}{\mathrm{dt}}G^*(t)\mathrm{dt}$ denote the optimal solution's total cost. We will define a potential function $\Phi(t)$ that will satisfy the following conditions:

1. **Boundary Condition**: Φ is 0 before any request arrives and 0 after all requests complete
2. **Arrival/Completion Condition**: Φ does not increase when a request is completed by LAPS or OPT or when a request arrives.
3. **Running Condition**: At all times t it is the case that $\frac{\mathrm{d}}{\mathrm{dt}}G(t) + \frac{\mathrm{d}}{\mathrm{dt}}\Phi \leq c\frac{\mathrm{d}}{\mathrm{dt}}G^*(t)$ where c is some positive constant.

By integrating the running condition over time, this is sufficient to show that our algorithm is c-competitive. This can be seen as follows. The second equality holds due to $\Phi(0) = \Phi(\infty) = 0$.

$$G = \int_0^\infty \frac{\mathrm{d}}{\mathrm{dt}}G(t)\mathrm{dt} = \int_0^\infty (\frac{\mathrm{d}}{\mathrm{dt}}G(t) + \frac{\mathrm{d}}{\mathrm{dt}}\Phi(t))\mathrm{dt} \leq \int_0^\infty c\frac{\mathrm{d}}{\mathrm{dt}}G^*(t)\mathrm{dt} \leq cG^*$$

Now we define our potential function. We assume without loss of generality that all requests arrive at distinct times, which will simplify the definition of the potential function. For a request $J_{p,i} \in N_a(t)$ let $\mathbf{rank}(J_{p,i}) = \sum_{J_{q,j} \in N_a(t), a_{q,j} \leq a_{p,i}} 1$ at time t be the number of jobs that have arrived during $[0, a_{p,i}]$ that are unsatisfied by the algorithm. Recall that $x_{p,i}(t)$ is the amount page p is broadcasted by our algorithm at time t due to $J_{p,i}$ and σ_p is the amount of page p that must be broadcasted after $a_{p,i}$ to satisfy $J_{p,i}$. Let $\mathtt{On}(p, i, t_1, t_2) = \int_{t_1}^{t_2} x_{p,i}(t)dt$. Let $y^*_p(t)$ be the amount of page p that is broadcasted at time t by OPT and let $\mathtt{Opt}(p, t_1, t_2) = \int_{t_1}^{t_2} y^*_p(t)dt$. We define the variable $z_{p,i}(t)$ for a request $J_{p,i}$ to be $\frac{\mathtt{On}(p,i,t,\infty)\mathtt{Opt}(p,a_{p,i},t)}{\sigma_p}$. Our potential function is,

$$\Phi(t) := \frac{1}{\epsilon} \sum_{J_{p,i} \in N_a(t)} \mathbf{rank}(J_{p,i}) \Big(\frac{z_{p,i}}{Q(\mathbf{rank}(J_{p,i}))} \Big)$$

Our potential function combines the potential functions of [8] and [28]. The potential function of [8] was used for broadcast scheduling without energy, which needs to somehow have the power function reflected in the potential if it is to work in our setting. To incorporate the power function we use some ideas given in [28].

As is usually the case, our potential is designed to approximate the algorithm future cost after subtracting the optimal solution's future cost. Our algorithm gives higher priority to jobs that have arrived recently. The

potential function is designed to capture the the remaining cost of the algorithm if it satisfies requests in the opposite order of arrival. Fix a request $J_{p,i}$. If the optimal solution satisfied request $J_{p,i}$ then the value of $z_{p,i}(t)$ should be thought of as the remaining volume of page p that must be broadcasted to satisfy request $J_{p,i}$. Assume $J_{p,i}$ has the highest rank in $N_a(t)$. Then $Q(\mathbf{rank}(J_{p,i}))$ is the speed that will be used at time t. The value of $\mathbf{rank}(J_{p,i})$ is the number of requests waiting for request $J_{p,i}$ to be satisfied. Thus, if requests are satisfied in opposite order of arrival, $\mathbf{rank}(J_{p,i})\Big(\frac{z_{p,i}}{Q(\mathbf{rank}(J_{p,i}))}\Big)$ is an estimate on the flow time that will be accumulated while the algorithm satisfies $J_{p,i}$ because it will take at least $\frac{z_{p,i}}{Q(\mathbf{rank}(J_{p,i}))}$ time units to satisfy request $J_{p,i}$.

3.1 Change of the Potential Function

It is easy to see that the potential function is not affected when the optimal solutions completes a request. Also the potential function does not increase when a request $J_{p,i}$ arrives since $z_{p,i} = 0$. Further, the potential is 0 before all requests arrive and after all requests are completed. We now show that Φ does not increase when a request is satisfied by the algorithm. This lemmas, combined with the previous arguments shows that Φ satisfies the boundary and arrival/completion conditions.

Lemma 3.1. *Φ does not increase when the algorithm completes a request.*

Proof. Consider a time t where that algorithm completes a request $J_{p,i}$. We can assume that $\mathbf{rank}(J_{p,i}) < |N_a(t)|$; otherwise, trivially there is no increase in Φ. The change in Φ is,

$$\Delta\Phi(t) = \frac{1}{\epsilon} \sum_{J_{p',j} \in N_a(t), a_{p',j} > a_{p,i}}^{|N_a(t)|} \Big(\frac{(\mathbf{rank}(J_{p',j}) - 1) z_i}{Q(\mathbf{rank}(J_{p',j}) - 1)} - \frac{\mathbf{rank}(J_{p',j}) z_i}{Q(\mathbf{rank}(J_{p',j}))} \Big)$$

To show that $\Delta\Phi(t) \leq 0$, we need to show that

$$\begin{aligned} \sum_{J_{p',j} \in N_a(t), a_{p',j} > a_{p,i}}^{|N_a(t)|} \frac{(\mathbf{rank}(J_{p',j}) - 1) z_i}{Q(\mathbf{rank}(J_{p',j}) - 1)} &\leq \sum_{J_{p',j} \in N_a(t), a_{p',j} > a_{p,i}}^{|N_a(t)|} \frac{\mathbf{rank}(J_{p',j}) z_i}{Q(i)} \Rightarrow \\ \sum_{J_{p',j} \in N_a(t), a_{p',j} > a_{p,i}}^{|N_a(t)|} \frac{(\mathbf{rank}(J_{p',j}) - 1)}{\mathbf{rank}(J_{p',j})} &\leq \sum_{J_{p',j} \in N_a(t), a_{p',j} > a_{p,i}}^{|N_a(t)|} \frac{Q(\mathbf{rank}(J_{p',j}) - 1)}{Q(\mathbf{rank}(J_{p',j}))} \end{aligned}$$

This is true by definition of Q and Proposition 2.3 □

Now we concentrate on the running condition, the final property of Φ that needs to be shown. Fix a time t. Let $N_o(t)$ be the set of requests unsatisfied by OPT at time t. First lets consider the change in $\Phi(t)$ due to the algorithm's processing. Recall that the algorithm broadcasts page p at a rate of $\frac{(1+6\epsilon)Q(|N_a(t)|)}{\beta|N_a(t)|}$ due to a request $J_{p,i} \in N'_a(t)$. Further, notice that $\mathtt{Opt}(p, a_{p,i}, t) \geq \sigma_p$ for any request $J_{p,i} \in N_a(t) \setminus N_o(t)$, since OPT must broadcast σ_p units of page p after $a_{p,i}$ to satisfy page p. Therefore, for any $J_{p,i} \in N'_a(t) \setminus N_o(t)$ we have $\frac{\mathrm{d}}{\mathrm{dt}} z_{p,i}(t) \geq \frac{\mathrm{d}}{\mathrm{dt}} \mathtt{On}(p, i, t, \infty) = -\frac{(1+6\epsilon)Q(|N_a(t)|)}{\beta|N_a(t)|}$. Notice that the $\mathbf{rank}$ of each request the algorithm is currently working on is at least $(1-\beta)N_a(t)$. This is because $N'_a(t)$ consists of the $\beta|N_a(t)|$ requests with latest arrival times. Using this, we can upperbound the change in $\Phi(t)$ due to the algorithm's processing,

$$\frac{\mathrm{d}}{\mathrm{dt}}\Phi \leq -\frac{1}{\epsilon} \sum_{J_{p,i} \in N'_a(t) \setminus N_o(t)} \mathbf{rank}(J_{p,i}) \frac{(1+6\epsilon)Q(|N_a(t)|)}{\beta N_a(t) Q(|N_a(t)|)} \leq - \sum_{J_{p,i} \in N'_a(t) \setminus N_o(t)} \frac{(1+6\epsilon)(1-\beta)}{\epsilon\beta} \quad (1)$$

Next, we consider the change in Φ due to the adversary's processing. Let p^* be the page which the adversary broadcasts. Let $s_o(t)$ be the speed the optimal solution processes page p^* at. Let $P^*(t) = P(s_o(t))$ be the power OPT uses at time t. The adversary can affect $z_{p^*,i}$ for any request $J_{p^*,i} \in N_a(t)$. We first observe the following,

$$\frac{\mathrm{d}}{\mathrm{dt}} \sum_{J_{p^*,i} \in N_a(t)} z_{p^*,i}(t) \leq \sum_{J_{p^*,i} \in N_a(t)} \frac{\mathtt{On}(p^*, i, t, \infty)}{\sigma_{p^*}} \cdot \Big(\frac{\mathrm{d}}{\mathrm{dt}} \mathtt{Opt}(p^*, a_{p^*,i}, t) \Big) \leq \frac{\mathrm{d}}{\mathrm{dt}} \mathtt{Opt}(p^*, a_{p^*,i}, t) = s_o(t) \mathrm{dt}$$

The last inequality holds since $\sum_{i, J_{p^*,i} \in N_a(t)} \mathtt{On}(p^*, i, \infty) \leq \sigma_{p^*}$. This is because the algorithm needs to only broadcast page p^* for a total of σ_{p^*} amount of time to satisfy all outstanding requests for page p^*. Let $J_{p^*,k}$ be the request in $N_a(t)$ such that $\frac{\mathbf{rank}(J_{p^*,k})}{Q(\mathbf{rank}(J_{p^*,k}))}$ is maximized. We can upper bound the increase in Φ due to OPT' processing as,

$$\frac{\mathrm{d}}{\mathrm{dt}} \Phi = \frac{1}{\epsilon} \Big(\frac{\mathbf{rank}(J_{p^*,k})}{Q(\mathbf{rank}(J_{p^*,k}))} \Big) \frac{\mathrm{d}}{\mathrm{dt}} \sum_{J_{p^*,i} \in N_a(t)} z_{p^*,i}(t) \leq \frac{\mathbf{rank}(J_{p^*,k}) s_o(t)}{\epsilon Q(\mathbf{rank}(J_{p^*,k}))}$$

Our goal is to show that $\frac{\mathrm{d}}{\mathrm{dt}} G(t) + \frac{\mathrm{d}}{\mathrm{dt}} \Phi(t) \leq \frac{2}{\epsilon^2} \frac{\mathrm{d}}{\mathrm{dt}} G^*(t)$. First we consider the case when the adversary uses power at least $|N_a(t)|$. In this case, the increase in the algorithm's objective can be charged directly to the optimal solution, along with any increase in the potential function.

Lemma 3.2. *If $Q(|N_a(t)|) \leq s_o(t)$ (equivalently, $P^*(t) \geq |N_a(t)|$) then $\frac{\mathrm{d}}{\mathrm{dt}} G(t) + \frac{\mathrm{d}}{\mathrm{dt}} \Phi(t) \leq \frac{2}{\epsilon} \frac{\mathrm{d}}{\mathrm{dt}} G^*(t)$.*

Proof. First we bound the increase in $\Phi(t)$ due to OPT's processing. Intuitively, the increase in OPT's objective, due to using a lot of power, is large enough to absorb the increase in Φ and the increase in algorithm's objective. By increasing Φ now and charging it to OPT, we can later use the decrease in Φ to pay for increases in the algorithm's objective. Recall that $J_{p^*,k}$ is request in $N_a(t)$ for page p^* that maximizes $\frac{\mathbf{rank}(J_{p^*,k})}{Q(\mathbf{rank}(J_{p^*,k}))}$. Let $\alpha |N_a(t)| = \mathbf{rank}(J_{p^*,k})$ and let $\gamma = \frac{s_o(t)}{Q(|N_a(t)|)}$. Notice that $\alpha \leq 1$ and $\gamma \geq 1$. The function Q is concave. Therefore, $Q(\mathbf{rank}(J_{p^*,k})) \geq \alpha Q(|N_a(t)|)$ by Proposition 2.3. The increase in $\Phi(t)$ due to OPT's processing can be bounded as follows.

$$\Big(\frac{\mathbf{rank}(J_{p^*,k})}{\epsilon} \Big) \frac{s_o(t)}{Q(\mathbf{rank}(J_{p^*,k}))} \leq \frac{1}{\epsilon} \alpha |N_a(t)| \gamma \frac{Q(|N_a(t))|}{Q(\mathbf{rank}(J_{p^*,k}))} \leq \frac{1}{\epsilon} \alpha |N_a(t)| \gamma \frac{Q(|N_a(t))|}{\alpha Q(|N_a(t)|)} \leq \frac{\gamma}{\epsilon} |N_a(t)|$$

The total power used by OPT is at least $P^* \geq \gamma |N_a(t)|$ since P is convex and the speed OPT uses is $\gamma Q(|N_a(t)|)$. Hence, $\frac{\mathrm{d}}{\mathrm{dt}} G^*(t) \geq \gamma |N_a(t)|$. Recall that $\frac{\mathrm{d}}{\mathrm{dt}} G(t) = 2|N_a(t)|$. Knowing that $\epsilon \leq \frac{1}{6}$ we have that,

$$\frac{\mathrm{d}}{\mathrm{dt}} G(t) + \frac{\mathrm{d}}{\mathrm{dt}} \Phi(t) \leq 2|N_a(t)| + \frac{\gamma}{\epsilon} |N_a(t)| \leq \frac{2\gamma}{\epsilon} |N_a(t)| \leq \frac{2}{\epsilon} \frac{\mathrm{d}}{\mathrm{dt}} G^*(t)$$

□

Due to the previous lemma, for the rest of this paper we can concentrate on the case where $Q(|N_a(t)|) > s_o(t)$. We begin by bounding the increase in $\Phi(t)$ due to OPT's processing using this assumption,

$$\begin{aligned}
\Big(\frac{\mathbf{rank}(J_{p^*,k})}{\epsilon}\Big)\frac{s_o(t)}{Q(\mathbf{rank}(J_{p^*,k}))} &\leq \Big(\frac{\mathbf{rank}(J_{p^*,k})}{\epsilon}\Big)\frac{Q(|N_a(t)|)}{Q(\mathbf{rank}(J_{p^*,k}))} \\
&\leq \Big(\frac{\mathbf{rank}(J_{p^*,k})}{\epsilon}\Big)\frac{(|N_a(t)|/\mathbf{rank}(J_{p^*,k}))Q(\mathbf{rank}(J_{p^*,k}))}{Q(\mathbf{rank}(J_{p^*,k}))} \\
&\leq \frac{1}{\epsilon}|N_a(t)| \qquad (2)
\end{aligned}$$

The first inequality holds since we assumed that $Q(|N_a(t)|) > s_o(t)$. The second inequality follows from definition of Q and Proposition 2.3. We can now prove the final case of the running condition.

Lemma 3.3. *If $s_o(t) < Q(|N_a(t)|)$ and $\beta = \epsilon$ then $\frac{\mathrm{d}}{\mathrm{dt}}G(t) + \frac{\mathrm{d}}{\mathrm{dt}}\Phi(t) \leq \frac{2}{\epsilon^2}\frac{\mathrm{d}}{\mathrm{dt}}G^*(t)$.*

Proof. We know that $\frac{\mathrm{d}}{\mathrm{dt}}G(t) = 2|N_a(t)|$. The increase in $\Phi(t)$ due to OPT's processing is at most $\frac{1}{\epsilon}|N_a(t)|$ and the change in $\Phi(t)$ due to the algorithm's processing is at most $-\sum_{J_{p,i}\in N'_a(t)\setminus N_o(t)}\frac{(1+6\epsilon)(1-\beta)}{\epsilon\beta}$. Combining these, we have

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{dt}}G(t) + \frac{\mathrm{d}}{\mathrm{dt}}\Phi(t) &\leq 2|N_a(t)| + \frac{1}{\epsilon}|N_a(t)| - \sum_{J_{p,i}\in N'_a(t)\setminus N_o(t)}\frac{(1+6\epsilon)(1-\beta)}{\epsilon\beta} \\
&\leq 2|N_a(t)| + \frac{1}{\epsilon}|N_a(t)| - \Big(\sum_{J_{p,i}\in N'_a(t)}\frac{(1+6\epsilon)(1-\beta)}{\epsilon\beta} - \sum_{J_{p,i}\in N_o(t)}\frac{(1+6\epsilon)(1-\beta)}{\epsilon\beta}\Big) \\
&\leq 2|N_a(t)| + \frac{1}{\epsilon}|N_a(t)| - \Big(\frac{(1+6\epsilon)(1-\beta)}{\epsilon}|N_a(t)| - \frac{(1+6\epsilon)(1-\beta)}{\epsilon\beta}|N_o(t)|\Big) \\
&\leq \frac{2}{\epsilon^2}|N_o(t)| \leq \frac{2}{\epsilon^2}\frac{\mathrm{d}}{\mathrm{dt}}G^*(t)
\end{aligned}$$

The second to last inequality follows from $\beta = \epsilon \leq \frac{1}{6}$. The last inequality follows since the optimal solution's flow time increases by $|N_o(t)|$ at time t. □

Combing Lemmas (3.2) and (3.3) along with setting β to be ϵ gives the running condition, namely $\frac{\mathrm{d}}{\mathrm{dt}}G(t) + \frac{\mathrm{d}}{\mathrm{dt}}\Phi(t) \leq \frac{2}{\epsilon^2}\frac{\mathrm{d}}{\mathrm{dt}}G^*(t)$. Thus, we have shown all the properties of Φ, which gives the following theorem.

Theorem 3.4. *The algorithm with $\beta = \epsilon \leq \frac{1}{6}$ is $(1+6\epsilon)$-speed $\frac{2}{\epsilon^2}$-competitive for flow plus energy in fractional broadcast scheduling.*

By using the reduction from integral to fractional broadcast scheduling in Corollary (2.5) and scaling ϵ, we have proven Theorem 1.1.

4 Conclusion

In this paper we initiated the study of energy in the broadcast scheduling model. We showed a scalable algorithm for average flow time plus energy. It is important to note that the algorithm BLAPS explicitly depends on the speed ϵ. That is, β depends on the speed ϵ. Recently, many algorithms developed for scheduling have this dependency [16, 24, 27]. It would be interesting to find an algorithm which does not depend on ϵ or show no such algorithm exists. It would also be of interest to consider objective functions that include energy minimization for the broadcast setting.

References

[1] S. Acharya, M. Franklin, and S. Zdonik. Dissemination-based data delivery using broadcast disks. *Personal Communications, IEEE [see also IEEE Wireless Communications]*, 2(6):50–60, Dec 1995.

[2] For Multihop Ad, Ya Xu, John Heidemann, and Deborah Estrin. Adaptive energy-conserving routing. Technical report, Research Report 527, USC/Information Sciences Institute, 2000.

[3] Demet Aksoy and Michael J. Franklin. "rxw: A scheduling approach for large-scale on-demand data broadcast. *IEEE/ACM Trans. Netw.*, 7(6):846–860, 1999.

[4] Susanne Albers and Hiroshi Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Transactions on Algorithms*, 3(4), 2007.

[5] Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Speed scaling with an arbitrary power function. In *SODA*, pages 693–701, 2009.

[6] Nikhil Bansal, Moses Charikar, Sanjeev Khanna, and Joseph (Seffi) Naor. Approximating the average response time in broadcast scheduling. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 215–221, 2005.

[7] Nikhil Bansal, Don Coppersmith, and Maxim Sviridenko. Improved approximation algorithms for broadcast scheduling. *SIAM J. Comput.*, 38(3):1157–1174, 2008.

[8] Nikhil Bansal, Ravishankar Krishnaswamy, and Viswanath Nagarajan. Better scalable algorithms for broadcast scheduling. In *ICALP*, 2010.

[9] Nikhil Bansal, Kirk Pruhs, and Clifford Stein. Speed scaling for weighted flow time. *SIAM J. Comput.*, 39(4):1294–1308, 2009.

[10] Amotz Bar-Noy, Randeep Bhatia, Joseph (Seffi) Naor, and Baruch Schieber. Minimizing service and operation costs of periodic scheduling. *Math. Oper. Res.*, 27(3):518–544, 2002.

[11] Yair Bartal and S. Muthukrishnan. Minimizing maximum response time in scheduling broadcasts. In *SODA '00: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 558–559, 2000.

[12] Ho-Leung Chan, Jeff Edmonds, Tak Wah Lam, Lap-Kei Lee, Alberto Marchetti-Spaccamela, and Kirk Pruhs. Nonclairvoyant speed scaling for flow and energy. In *STACS*, pages 255–264, 2009.

[13] Ho-Leung Chan, Jeff Edmonds, and Kirk Pruhs. Speed scaling of processes with arbitrary speedup curves on a multiprocessor. In *SPAA*, pages 1–10, 2009.

[14] Jessica Chang, Thomas Erlebach, Renars Gailis, and Samir Khuller. Broadcast scheduling: algorithms and complexity. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 473–482. Society for Industrial and Applied Mathematics, 2008.

[15] Chandra Chekuri, Sungjin Im, and Benjamin Moseley. Longest wait first for broadcast scheduling. In *WAOA '09: Proceedings of 7th Workshop on Approximation and Online Algorithms*, 2009.

[16] Chandra Chekuri, Sungjin Im, and Benjamin Moseley. Minimizing maximum response time and delay factor in broadcasting scheduling. In *ESA '09: Proceedings of the seventeenth annual European symposium on algorithms*, pages 444–455, 2009.

[17] Benjie Chen, Kyle Jamieson, Hari Balakrishnan, and Robert Morris. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. In *ACM Wireless Networks Journal*, pages 85–96, 2001.

[18] R. K. Deb. Optimal control of bulk queues with compound poisson arrivals and batch service. *Opsearch.*, 21:227–245, 1984.

[19] R. K. Deb and R. F. Serfozo. Optimal control of batch service queues. *Adv. Appl. Prob.*, 5:340–361, 1973.

[20] Jeff Edmonds. Scheduling in the dark. *Theor. Comput. Sci.*, 235(1):109–141, 2000.

[21] Jeff Edmonds, Sungjin Im, and Benjamin Moseley. Online scalable scheduling for the ℓ_k-norms of flow time without conservation of work. Manuscript, 2010.

[22] Jeff Edmonds and Kirk Pruhs. Multicast pull scheduling: When fairness is fine. *Algorithmica*, 36(3):315–330, 2003.

[23] Jeff Edmonds and Kirk Pruhs. A maiden analysis of longest wait first. *ACM Trans. Algorithms*, 1(1):14–32, 2005.

[24] Jeff Edmonds and Kirk Pruhs. Scalably scheduling processes with arbitrary speedup curves. In *SODA '09: Proceedings of the Nineteenth Annual ACM -SIAM Symposium on Discrete Algorithms*, pages 685–692, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[25] Thomas Erlebach and Alexander Hall. Np-hardness of broadcast scheduling and inapproximability of single-source unsplittable min-cost flow. *J. Scheduling*, 7(3):223–241, 2004.

[26] Rajiv Gandhi, Samir Khuller, Srinivasan Parthasarathy, and Aravind Srinivasan. Dependent rounding and its applications to approximation algorithms. *J. ACM*, 53(3):324–360, 2006.

[27] Anupam Gupta, Sungjin Im, Ravishankar Krishnaswamy, Benjamin Moseley, and Kirk Pruhs. Scheduling jobs with varying parallelizability to reduce variance. In *SPAA '10: 22nd ACM Symposium on Parallelism in Algorithms and Architectures*, 2010.

[28] Anupam Gupta, Ravishankar Krishnaswamy, and Kirk Pruhs. Scalably scheduling power-heterogeneousprocessors. In *ICALP (1)*, 2010.

[29] Alexander Hall and Hanjo Täubig. Comparing push- and pull-based broadcasting. or: Would "microsoft watches" profit from a transmitter?. In *Proceedings of the 2nd International Workshop on Experimental and Efficient Algorithms (WEA 03)*, pages 148–164, 2003.

[30] Sungjin Im and Benjamin Moseley. An online scalable algorithm for average flow time in broadcast scheduling. In *SODA '10: Proceedings of the Twentieth Annual ACM -SIAM Symposium on Discrete Algorithms*, 2010.

[31] Sandy Irani, Sandeep K. Shukla, and Rajesh Gupta. Algorithms for power savings. *ACM Transactions on Algorithms*, 3(4), 2007.

[32] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000.

[33] Bala Kalyanasundaram, Kirk Pruhs, and Mahendran Velauthapillai. Scheduling broadcasts in wireless networks. *Journal of Scheduling*, 4(6):339–354, 2000.

[34] Tak Wah Lam, Lap-Kei Lee, Isaac Kar-Keung To, and Prudence W. H. Wong. Speed scaling functions for flow time scheduling based on active job count. In *ESA*, pages 647–659, 2008.

[35] Kirk Pruhs, Patchrawat Uthaisombut, and Gerhard J. Woeginger. Getting the best response for your erg. *ACM Transactions on Algorithms*, 4(3), 2008.

[36] J. Weiss. Optimal control of batch service queues with nonlinear waiting costs. *Modeling and Simulation*, 10:305–309, 1979.

[37] J. Weiss and S. Pliska. Optimal policies for batch service queueing systems. *Opsearch*, 19(1):12–22, 1982.

[38] J. Wong. Broadcast delivery. *Proceedings of the IEEE*, 76(12):1566–1577, 1988.

A Reduction of Integral to Fractional Broadcast Scheduling

In this section we prove Theorem 2.4. Consider any sequence of requests and let $\mathcal{S}$ denote a valid fractional broadcast schedule. We now define an algorithm to construct an integral schedule $\mathcal{S}'$. In the integral schedule we will use $0 < \epsilon \leq 1$ resource augmentation over the schedule $\mathcal{S}$. Recall that in an integral schedule, a request $J_{p,i}$ must receive σ_p unit sized pieces of page p in *sequential* order. We will assume that the fractional schedule works on at most one page during a unit time slot. We can assume this without loss of generality because we can set a unit time slot to be arbitrarily small, since we are assuming preemption. Let $f_{p,i}$ denote the time request $J_{p,i}$ is fractionally satisfied in $\mathcal{S}$. Let $f^I_{p,i}$ denote the time that $J_{p,i}$ is integrally satisfied in $\mathcal{S}'$. Our algorithm and analysis build on the reduction from fractional to integral broadcast scheduling given in [8] where broadcasts are always sent at a fixed speed. Since the processor speeds can vary in our setting, in our analysis we will have to be careful about how speed is distributed and how processing power is accounted for.

Our algorithm will keep track of a queue $\mathcal{Q}$ of tuples. A tuple will be of the form $\langle p, w, b, k\rangle$. Here p corresponds to a page. The value of $k \in \mathbb{R}$ will correspond to the part of page p that will be broadcasted. The variable $w \in \mathbb{R}$ will be called the *width* and $b \in \mathbb{R}$ will be the *start time*. Each tuple $\tau = \langle p, w, b, k\rangle$ will correspond to some request $J_{p,i}$ and the width will be $w = f_{p,i} - a_{p,i}$. This request will determine the part of page p that will be broadcasted and the speed of the broadcast. The request corresponding to a tuple could be updated. However, after the first piece of the page is broadcasted, the request corresponding to the tuple will not be changed.

At any time when not broadcasting a page, our algorithm will determine a tuple $\tau = \langle p, w, b, k\rangle$ to broadcast. Let $J_{p,i}$ correspond to this tuple. Let $s(p, i, k)$ denote the speed used by $\mathcal{S}$ in the kth time slot devoted to page p after time $a_{p,i}$. Notice that a $s(p, i, k)$ volume of page p is broadcasted in this time slot under $\mathcal{S}$. When broadcasting the tuple τ, we mean that our algorithm is broadcasting the page p with speed $(1 + \epsilon)s(p, i, k)$. Our algorithm stops broadcasting p once a $s(p, i, k)$ volume of work is completed. Notice that this takes $\frac{1}{1+\epsilon}$ time because our algorithm runs at speed $(1 + \epsilon)s(p, i, k)$ and $\mathcal{S}$ took one time unit to broadcast the $s(p, i, k)$ volume of page p. If $k = 1$ then the algorithm broadcasts p from the beginning of the page. Otherwise, our algorithm broadcasts p from where it left off. Besides scaling the speeds, the algorithm is the same as that used in [8].

Algorithm: GenRounding(t)

All requests arrive unmarked
Simulate the schedule $\mathcal{S}$
for Any unmarked request $J_{p,i}$ completed by $\mathcal{S}$ at time t **do**
  if There is a tuple $\tau = \langle p, w, b, k\rangle \in \mathcal{Q}$ for page p where $b \geq a_{p,i}$ **then**
    Update the width of τ to be $w = \min\{w, f_{p,i} - a_{p,i}\}$
  else
    Insert the tuple $\langle p, (f_{p,i} - a_{p,i}), \infty, 1\rangle$ into $\mathcal{Q}$
  end if
end for
Dequeue the tuple $\tau = \langle p, w, b, k\rangle$ with minimum width, breaking ties arbitrarily
Let $J_{q,j}$ be the request which gave τ the width w
Broadcast the tuple τ
if This broadcast was the first unit piece of page p. That is, $k = 1$ **then**
  Set $b = t$
end if
if This broadcast was the last unit piece of page p. That is, $\sum_{i=1}^{k} s(q, j, k) = \sigma_p$ **then**
  Mark all requests for page p that arrived before time s
else
  Update τ to be $\langle p, w, b, k + 1\rangle$
end if

It can be observed that our algorithm is online if $\mathcal{S}$ is online. First we show that the power used by our algorithm is most the power used by $\mathcal{S}$.

Lemma A.1. *Consider any two tuples τ_j and τ_k for the same page q broadcast by $\mathcal{S}'$. Let $J_{q,j}$ and $J_{q,k}$ be the requests which gave these tuples their width and speed, respectively. Let $j > k$ then $a_{q,j} \geq f_{q,k}$*

Proof. The algorithm $\mathcal{S}'$ put the tuple τ_k into $\mathcal{Q}$ at time $f_{q,k}$. Hence, the start time $b(\tau_k)$ of τ_k must be after $f_{q,k}$. Since the algorithm inserted a new tuple for $J_{q,j}$, it must be that $b(\tau_k) < a_{q,j}$. Hence, $f_{q,k} \leq b(\tau_k) < a_{q,j}$. □

Consider any unit time slot in $\mathcal{S}$. The previous lemma implies that only one tuple broadcasted in $\mathcal{S}'$ will correspond to this time slot. This implies the following claim.

Claim A.2. *The power used by $\mathcal{S}'$ is at most the power used by $\mathcal{S}$.*

It remains to show that $f^I_{p,i} - a_{p,i} \leq \frac{2}{\epsilon}(f_{p,i} - a_{p,i}) + \frac{5}{\epsilon}$ for any request $J_{p,i}$. Fix a request $J_{p,i}$. If at time $f_{p,i}$ the request $J_{p,i}$ is marked then $f^I_{p,i} - a_{p,i} \leq \frac{2}{\epsilon}(f_{p,i} - a_{p,i}) + \frac{5}{\epsilon}$ and we are done. Thus, we assume that $J_{p,i}$ is unmarked at time $f_{p,i}$. Since, $J_{p,i}$ is unmarked, there exists a tuple $\tau = \langle p, w, b, k\rangle$ in $\mathcal{Q}$ at time t where $w \leq f_{p,i} - a_{p,i}$. Let t_A be the earliest time before time t where every tuple dequeued during $[t_A, f_{p,i}]$ has width less than w. Let t_B be the latest time after time t where every tuple dequeued during $[f_{p,i}, t_B]$ has width less than w. By definition of the algorithm, at each time during $[t_A, t_B]$ only tuples with width at most w are dequeued. Let $\prod$ denote the set of tuples broadcasted during $[t_A, t_B]$. Notice by definition of the algorithm, once a tuple is broadcasted, the width of the tuple never changes.

Claim A.3. *Consider any $\tau \in \prod$ and let $J_{q,j}$ be the request that gave τ its width. Then $a_{q,j} \geq t_A - w$.*

Proof. Since the tuple τ was broadcasted during $[t_A, t_B]$ the width $w(\tau)$ must be less than w. By definition of t_A there are no tuples with width less than w in $\mathcal{Q}$ at time $t_A - 1$. Thus, $f_{q,j} \geq t_A$ and by definition of width, $a_{q,j} \geq f_{q,j} - w \geq t_A - w$. □

Let $N = |\prod|$. All N tuples in $\prod$ correspond to some unit time slot where $\mathcal{S}$ preformed a broadcast. By the previous claim, all of these broadcasts occurred after time $t_A - w$. Further, by definition of the algorithm all of these broadcasts occurred before time t_B. This is because, a tuple corresponding to a request $J_{q,k}$ will not be inserted into $\mathcal{Q}$ until after time $f_{q,k}$ and all of the requests corresponding to tuples in $\prod$ are satisfied in $\mathcal{S}'$ by time t_B. This implies that the schedule $\mathcal{S}$ uses at least N time slots during $[t_A - w, t_B]$. Thus we have,

$$N \leq t_B - t_A + w + 2$$

Since our algorithm has ϵ resource augmentation, it takes $\frac{1}{1+\epsilon}$ time to broadcast one tuple. Since our algorithm is always busy during $[t_A, t_B]$ we have that

$$N \geq (1+\epsilon)(t_B - t_A) - 3$$

This implies that $t_B - t_A \leq \frac{w+5}{\epsilon}$. For the request $J_{p,i}$ we know that $a_{p,i} \geq t_A - w$ by Claim A.3. Thus, $t_B - a_{p,i} + w \leq \frac{w+5}{\epsilon}$. Knowing that $f^I_{p,i} \leq t_B$ and $\epsilon \leq 1$, we have $(f^I_{p,i} - a_{p,i}) \leq \frac{2w+5}{\epsilon}$. Combining this with Claim A.2 and the fact that our algorithm is online, we have Theorem 2.4.